DB2 Universal Database for z/OS

Version 8

What's New?

GC18-7428-01

Contents

Chapter 2. What's new in Version 8 of DB2 UDB for z/OS?

IBM delivers enhancements to DB2 UDB for z/OS that support your needs in the zSeries environment. Version 8 of DB2 UDB for z/OS delivers improvements in the following areas:

More flexibility with SQL

Version 8 greatly expands support for SQL functions in online transaction processing environments through the following enhancements:

Figure 1. Version 8 of DB2 UDB for z/OS gives you more flexibility with SQL.

SELECT FROM INSERT statement

Recent releases of DB2 have provided enhancements (such as ROWID columns, identity columns, and triggers) in which DB2 or a trigger, rather than an application program, inserts data into DB2 tables. Before Version 8, you could not immediately determine the values that were inserted into DB2 tables by mechanisms other than application programs. Now, in Version 8, you can select values from rows that are being inserted into a DB2 table by specifying the SELECT FROM INSERT statement. The rows that are inserted into the target table produce a result table; the columns of the result table can be referenced in the SELECT list of the query. When you insert one or more new rows into a table, you can retrieve the following values from the result table:

- The value of an automatically generated column such as a ROWID column or an identity column

- Any column values that are the result of an expression
- Any default values for columns
- All values for an inserted row, without specifying individual column names
- All values that are inserted by a multiple-row INSERT operation
- Values that are changed by a BEFORE INSERT trigger

Generation of unique sequential numbers for applications

In prior releases and in Version 8, identity columns are used to generate sequential numbers; however, an identity column is a column of a table and is therefore associated with the table. Version 8 of DB2 UDB for z/OS introduces a new SQL data object, *sequence,* that provides recoverable, unique sequential numbers for applications. This data object also enhances the portability of your applications across the DB2 family and other vendor operating systems. In contrast to identity columns, sequences are stand-alone objects that applications can use to avoid concurrency and performance problems that can result when applications generate their own sequence numbers. After a sequence is defined, it can be concurrently accessed and incremented by many users, including multiple DB2 members in a data sharing group.

Ability to alter identity column attributes

Since identity columns were first introduced in Version 6, many users have asked for the ability to alter many of the attributes of identity columns. With Version 8, you can use the ALTER COLUMN clause of the ALTER TABLE statement to change all of the attributes of an identity column except the data type. The ability to alter attributes of identity columns eliminates the need to drop and re-create a table, which makes identity columns extremely flexible to use. For example, you can now alter the sequence attributes of existing identity columns, restart the column values from the new value, avoid minimum and maximum values, and control the order in which column values are generated.

Dynamic scrollable cursors

Version 7 provided the static scrollable cursor function in which scrolling is performed on a materialized global temporary table. Version 8 extends the scrollable cursor function by implementing a *dynamic scrollable cursor.* A dynamic scrollable cursor lets applications scroll directly on the base table while accessing the most current data, including newly inserted rows. The dynamic scrollable cursor function is particularly beneficial for large result sets that would otherwise need to be materialized. Dynamic scrolling is also supported by data-partitioned secondary indexes, index scans, and table space scans.

Scalar fullselects in SQL statements

Version 8 of DB2 UDB for z/OS adds more power to your queries and gives you more flexibility with your applications across the DB2 family. DB2 now supports scalar fullselects (with some restrictions) wherever expressions are allowed in SQL statements. Each scalar fullselect within an SQL statement returns a single row that consists of a single column.

Integrated XML publishing functions in DB2

In Version 8, DB2 UDB for z/OS enhances its leadership as an enterprise database server by providing a set of SQL built-in functions that allow applications to generate XML data from relational data with high performance. The XML

publishing functions can reduce application development efforts in generating XML data for data integration, information exchange, and Web services. With XML publishing functions, DB2 can:

- Generate XML elements with optional attributes from columns and expressions
- Generate XML data with hierarchical structures, through grouping and concatenation, for data that has parent-child relationships

Common table expressions in SQL statements

Version 8 provides improved usability and consistency across the DB2 family through support for common table expressions in SQL statements. A common table expression is like a temporary view that is defined and used for the duration of an SQL statement. You can reference each common table expression many times in an SQL statement; all references to a common table expression share the same result table. In contrast, regular views or nested table expressions are derived each time that they are referenced. Common table expressions can also improve performance in some cases because values are computed once rather than several times.

Recursive SQL

In Version 8, you can use common table expressions to create recursive SQL. If a fullselect of a common table expression contains a reference to itself in a FROM clause, the common table expression is known as a *recursive common table expression.* Common table expressions and recursive SQL improve usability and consistency within the DB2 UDB family. In some cases, common table expressions and recursive SQL can be used to improve performance because values are derived once rather than several times. Queries that use recursion are useful in applications like bill-of-materials applications, network planning applications, and reservation systems.

CURRENT PACKAGE PATH special register

Support for a new special register, CURRENT PACKAGE PATH, reduces network traffic, simplifies application coding, and improves processing time and elapsed time for stored procedures, user-defined functions, Java programs that use SQLJ, and applications that use DRDA from a z/OS requester. Many installations use more than one collection for packages. In prior releases, applications that do not use plans must issue the SET CURRENT PACKAGESET statement each time a package from a different collection is used.

With support for the CURRENT PACKAGE PATH special register in Version 8, an application programmer can specify a list of package collections in one SET CURRENT PACKAGE PATH statement. The new SET CURRENT PACKAGE PATH special register also lets you implement a nested procedure or a user-defined function, regardless of your run-time environment, and lets you specify multiple collections.

GET DIAGNOSTICS statement

The GET DIAGNOSTICS statement returns diagnostic information about the previous SQL statement that was executed, and is consistent with the ANSI/ISO Core Level SQL Standard for 1999. The GET DIAGNOSTICS statement in Version 8 is more robust and less restrictive than the SQLCA. You can use the GET DIAGNOSTICS statement to request information about:

- The entire SQL statement. In addition, the GET DIAGNOSTICS statement can return longer names.

- Conditions, including multiple conditions for multiple-row statements, and the error message that is associated with an individual error. GET DIAGNOSTICS supports SQL error message tokens that are greater than 70 bytes.
- Connections, if the SQL statement was a CONNECT statement.

You can issue the GET DIAGNOSTICS statement from an embedded application and from within an SQL procedure.

Compare null values with the DISTINCT predicate

The new DISTINCT predicate lets you compare null values and simplifies the SQL that you need to write when you need to find values that might be null. Two forms of the DISTINCT predicate are:

IS DISTINCT FROM
Creates an expression in which both values are not equal or one value is null.

IS NOT DISTINCT FROM
Creates an expression in which one value is equal to another value, or both values are null.

Improved security

Version 8 provides new options for e-business and high security with multilevel security and row level security. Together, these improvements let you identify system users more easily and significantly increase the granularity of your security authorizations. These improvements result in additional flexibility for applications and SQL. Other new features, such as encryption, also improve security. The following sections provide more details about security improvements in Version 8:

New built-in functions for data encryption and decryption

Version 8 of DB2 UDB for z/OS provides new built-in functions for data encryption and decryption that let you protect valuable data as it is stored in or retrieved from a DB2 subsystem. The ENCRYPT function lets you encrypt and store data in columns of DB2 tables. You can copy, restore, or move encrypted data between DB2 subsystems.

DB2 gives you the flexibility to encrypt all values in a column of data with the same (or *common*) password or to allow many different passwords within a column. For example, you can set up a common password that lets users access a specific view of the data. Or you can let individual users create their own passwords when they set up an account with their credit card numbers and related information on an e-business Web site.

Additional built-in functions provide support for generating unique sequential values in a table, decrypting encrypted data, and setting up password protection.

New data encryption tool

IBM Data Encryption for IMS and DB2 Databases, one of the IBM DB2 and IMS Tools products, is a single tool that you can use to protect sensitive IMS and DB2 UDB for z/OS data. In DB2, data encryption and decryption is implemented through the standard EDITPROC exit routine. The exit code uses the zSeries and S/390® Crypto Hardware to encrypt data for storage and decrypt data for application use. This tool can help you save the time and effort that is required to write and maintain your own encryption software. For additional information about IBM Data Encryption for IMS and DB2 Databases, see www.ibm.com/software/data/db2imstools/

Multilevel security with row-level granularity

Multilevel security is a security policy that lets you classify data and users based on a system of hierarchical security levels that is combined with a system of nonhierarchical security categories. The goals of multilevel security are twofold: To prevent individuals from accessing information that is classified at a level that is higher than their authorization allows, and to prevent individuals from declassifying information.

Version 8 of DB2 UDB for z/OS supports multilevel security with row-level granularity, which lets you restrict individual user access to a specific set of rows in a table. Multilevel security with row-level granularity offers several advantages over current authorization techniques:

- Security enforcement is mandatory and automatic; a user is checked at run time. This technique complements existing discretionary checks.
- You can perform security checks that are difficult to express through traditional SQL views or queries.
- Multilevel security does not rely on special views or database variables to provide row-level security control.
- Security controls are consistent and integrated across the system so that you can avoid defining users, objects, access, and security labels more than once. Access to files, database, printers, terminals, and other resources can have a single security control point.

Easier identification of system users

Prior to Version 8, if you have a large server, such as WebSphere Application Server, connected to DB2, all the threads for a connection show a single identity (the server). Now, Version 8 provides four new special registers that you can use in your applications to more easily identify system users.

CLIENT_ACCTNG special register

The CLIENT_ACCTNG special register contains the current value of the accounting string from the client information that is specified for a connection. Now you can get the current value of the accounting string that is used in a specific connection. For example:

```
SET :ACCT_STRING = CLIENT_ACCTNG
```

CLIENT_APPLNAME special register

The CLIENT_APPLNAME special register contains the value of the application name from the client information that is specified for a connection. Now you can select which departments can use the application that is used in a specific connection. For example:

```
SELECT DEPT
  FROM DEPT_APPL_MAP
  WHERE APPL_NAME = CLIENT_APPLNAME
```

CLIENT_USERID special register

The CLIENT_USERID special register contains the value of the client user identifier from the client information that is specified for a connection. Now you can find out in which department a current client user ID works. For example:

```
SELECT DEPT
  FROM DEPT_USERID_MAP
  WHERE USER_ID = CLIENT_USERID
```

CLIENT_WRKSTNNAME special register

The CLIENT_WRKSTNNAME special register contains the value of the workstation name from the client information that is specified for a connection. Now you can get the workstation name that is used for a particular connection. For example:

```
SET :WS_NAME = CLIENT_WRKSTNNAME
```

Session variables

Support for session variables in Version 8 gives you another means by which to provide information to applications. DB2 sets some session variables, and you can set other session variables in the connection and signon exit routines. A new built-in function, GETVARIABLE, retrieves the values of a session variable. You can use this function to enforce security policies in views, triggers, stored procedures, and constraints. Application programmers and SQL users can now access the information that is set by DB2: Plan names, package names, DB2 version identifiers, security labels, and system CCSIDs. If you require more general, flexible controls for primary security, you can use session variable information to complement other security mechanisms.

Improved encrypted security in distributed computing environments

New Distributed Relational Database Architecture™ (DRDA®) security options provide the following data security improvements in distributed computing environments:

- DB2 UDB for z/OS servers can provide secure, high-speed data encryption and decryption.
- DB2 UDB for z/OS requesters now have the option of encrypting user IDs and optionally, passwords when they connect to remote servers. Requesters can also encrypt security-sensitive data when they communicate with servers, so that the data is secure when it travels over the network.

Enhanced compatibility with the DB2 family

The drive toward family compatibility continues in Version 8 with:

- "Extended limits for names and SQL statements" on page 11
- "Significant support for Unicode and long names in the DB2 catalog" on page 12
- "Network computing enhancements" on page 13
- "IBM z/OS Application Connectivity to DB2 for z/OS feature" on page 15
- "ODBC expands support for encoding schemes" on page 16

Most of the enhancements that make SQL more flexible, as discussed in "More flexibility with SQL" on page 5, also improve compatibility across the DB2 family.

The best reference for developing applications that are portable across the DB2 family is *IBM DB2 Universal Database SQL Reference for Cross-Platform Development*, available at www.ibm.com/software/db2zos/library.html

This book is updated as changes in SQL language elements are implemented across members of the DB2 family.

Extended limits for names and SQL statements

Version 8 of DB2 UDB for z/OS takes a giant leap over the current limits for column names, for SQL statements, for index keys and predicates, and for the number of tables that can be joined in a single FROM clause with the following enhancements:

Longer column names

In Version 8, the maximum length for table and view names is extended from 18 bytes to 128 bytes. The maximum length of column names is changed from 18 bytes to 30 bytes. Names for many other objects, such as indexes, statements, schemas, procedures, and triggers are also extended to 128 bytes.

Longer and more complex SQL statements

SQL statements can now be up to 2 MB in length. A number of the Version 8 capabilities stretch the limit on the size of an SQL statement. Long names and support for up to 4096 partitions require more space. An SQL procedure must be stated completely within a single SQL statement. Other changes in DB2 allow larger structures and, therefore, larger statements. SQL statements that are too large or too complex become very rare in the Version 8 environment.

Extensions to SQL procedure statements

In Version 8, the 2-MB extension to the length of an SQL statement also applies to the CREATE PROCEDURE statement. Specifically, the length of an individual SQL procedure statement, which consists of SQL control statements and SQL statements in the procedure body, is extended to 2 MB. As a result, if you specify an SQL control statement as the procedure body, you can include multiple SQL procedure statements within that control statement, each of which is now extended to 2 MB. This enhancement significantly increases the power and flexibility of SQL procedures.

Longer index keys and predicates

The maximum length of an index key is increased from 255 bytes to 2000 bytes. Similarly, the maximum length for a predicate operand is increased from 255 bytes to 32 704 bytes, which is the maximum length of a VARCHAR column.

Greater number of tables joined in a single FROM clause

Prior releases of DB2 let you join up to 15 tables in a single FROM clause. Many users need to run queries that join more than 15 tables. In some cases, users need to join as many as 80 tables; the trend is moving toward joining even more tables. Version 8 continues to meet user needs by offering support for joining up to 225 tables in a single FROM clause.

Fewer restrictions for column functions

The argument of a column function is a set of like values that is derived from an expression. Prior to Version 8 of DB2, the expression for the argument was required to include a reference to a column (which was referred to as a *column function*). In Version 8, you no longer need to specify a column name in the expression. Because a column reference is no longer required, column functions are now called *aggregate functions*.

Qualified columns in the INSERT statement

In prior releases of DB2, you cannot qualify the names of columns when you insert data into a column. In Version 8, you can use qualified column names in an INSERT statement just like you can in an UPDATE statement.

ORDER BY clause for the SELECT INTO statement

The SELECT INTO statement must produce a result that contains a single row. Prior to Version 8, you could specify the FETCH FIRST 1 ROW clause to ensure that only a single row was returned if the result set of the query could result in more than one row. However, you could not specify the ORDER BY clause to affect which row was returned. With Version 8, you can now specify ORDER BY. When you use both the FETCH FIRST 1 ROW and ORDER BY clauses, the result set is ordered first, and then the first row is returned.

Expressions in the GROUP BY clause

To enable greater portability of applications, Version 8 of DB2 UDB for z/OS lets you specify the same expressions in the GROUP BY clause that you can specify in HAVING, SELECT, and ORDER BY clauses. Now you can have family consistency without rewriting your SQL statements.

More than one DISTINCT keyword allowed in a single query

Version 8 enhancements to SQL support give you the flexibility to use more than one DISTINCT keyword in a single query. As a result, you no longer need to write multiple queries to retrieve multiple distinct column values.

Additional input format for timestamp strings

In addition to using a dash to separate the date portion and the time portion of a timestamp string, you can now use a blank as the separator. The ODBC and JDBC string representations of a timestamp use the format in which the blank is the separator.

Explicitly defined ROWID columns are not required for large objects

In Version 8, you no longer need to explicitly define a ROWID column when you define a large object (LOB) column. If a ROWID column does not exist when you define a LOB column with either the ALTER TABLE or CREATE TABLE statement, DB2 implicitly generates a ROWID column.

Descriptions for plans and packages in the DB2 catalog

You can now provide descriptions for plans and packages in the DB2 catalog. Support for comments for plans and packages simplifies documenting and tracking your objects and increases compatibility within the DB2 UDB family.

Implicit drop of declared global temporary tables at commit

In Version 8, you can specify that DB2 is to implicitly drop declared global temporary tables at a commit operation. The new ON COMMIT DROP TABLE clause of the DECLARE GLOBAL TEMPORARY TABLE statement lets you drop the declared global temporary table at commit if no open cursors on the table are defined as WITH HOLD. This enhancement is particularly important for distributed applications and stored procedures because cleanup can occur when cursors are closed.

Significant support for Unicode and long names in the DB2 catalog

Architectural changes in Version 8 expand the DB2 catalog for long names and Unicode.

Unicode

With Version 8 comes significant support for Unicode, which means that you can manage data from around the world. DB2 now converts any SQL statement to Unicode before parsing; as a result, all characters parse correctly. DB2 also supports hexadecimal string constants.

Long names

The following enhancements for long names in Version 8 make DB2 UDB for z/OS compatible with other members of the DB2 family:

- Longer string constants (up to 32 704 bytes)
- Longer index keys (up to 2000 bytes)
- Longer predicates (up to 32 704 bytes)
- Longer object names (up to 30 characters for column names and up to 128 characters for most other SQL objects)

Unicode, EBCDIC, and ASCII columns in the same SQL statement

Prior to Version 7, DB2 supported a limited set of coded character set identifiers (CCSIDs) to store data in EBCDIC and ASCII encoding schemes. Version 7 of DB2 for z/OS and OS/390 introduced the Unicode encoding scheme to address the problems that are encountered when users who live in different geographies and who speak many different languages interact with the same DB2 server. The Unicode encoding scheme represents the characters of many different geographies and languages. With Version 7 support for the Unicode encoding scheme, you cannot reference table objects that are defined with different encoding schemes in the same SQL statement.

Version 8 further expands DB2 UDB for z/OS support of multiple encoding schemes by letting you reference tables or table functions with different CCSIDs in the same SQL statement. Support for multiple CCSIDs gives you the flexibility to join tables that have Unicode, EBCDIC, and ASCII columns.

Network computing enhancements

Version 8 of DB2 UDB for z/OS provides the following enhancements for network computing:

- "Increased portability of applications through consistent access to DB2 family servers"
- "Improved performance for remote queries" on page 14
- "Improvements in connectivity" on page 15

Increased portability of applications through consistent access to DB2 family servers

In prior releases, differences in access paths for applications that run on DB2 UDB for Linux, UNIX® and Windows® and on DB2 UDB for z/OS required duplication of effort in developing and testing applications for the different run-time environments. In addition, accessing a DB2 UDB for Linux, UNIX and Windows server and a DB2 UDB for z/OS server required different database connection protocols. Each connection protocol, in turn, defined a different set of methods to implement the same functions.

Enhancements in Version 8 remove roadblocks to performance and DB2 family compatibility by providing support for a common client and standardizing database connection protocols based on the Open Group Technical Standard DRDA Version 3. Three components comprise the common client: A C common client for ODBC, a Java common client for SQLJ and JDBC, and an administrative client.

Together, these components provide consistent access to servers across DB2 environments through a common run-time environment that has a single access path for all applications. (See Figure 2.) As a result, new function and new applications can go into production more quickly because you can write an application once and use it in any of the DB2 common client environments.

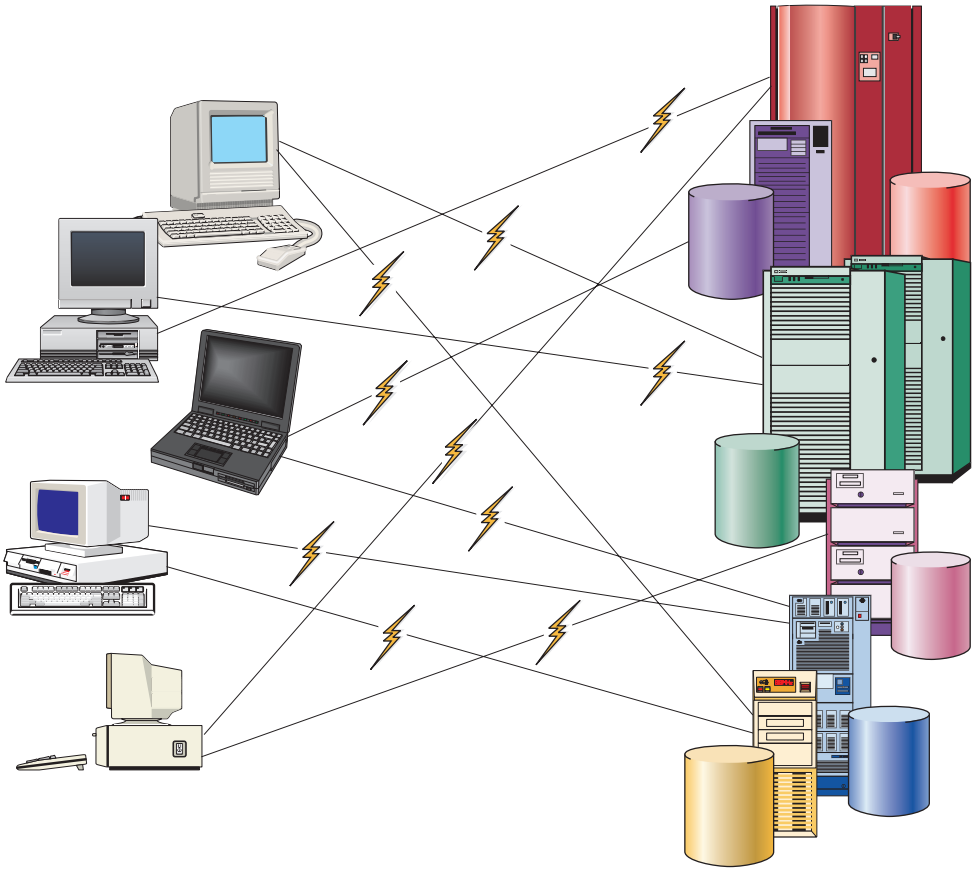

Figure 2. The DB2 common clients offer increased portability of your applications through consistent access to DB2 family servers.

Improved performance for remote queries

Version 8 provides the following new function to improve performance for remote queries:

- Version 8 provides a new server interface to boost the performance of processing remote fetches.
- Multiple-row fetch and multiple-row insert functions improve the DRDA interface to DB2 and provide a more efficient approach for generating query blocks. DB2 builds the query directly into the communication buffer in a single request, which minimizes the number network operations that are required to generate a single query block.
- Improvements to DRDA support in DB2 improve the performance of read-only, single-row cursors.
- The SQL cancel function allows a JDBC or CLI application to cancel long-running requests on a DB2 UDB for z/OS server.

Improvements in connectivity

As a result of support for Version 3 of Open Group Technical Standard DRDA, the following connectivity improvements in Version 8 of DB2 UDB for z/OS are available:

- A DB2 database for Linux, UNIX, and Windows is known in the network by its database name. Applications use the database name to connect to an instance of a DB2 UDB for Linux, UNIX and Windows database. When DB2 is deployed to a large number of servers in a Linux, UNIX, or Windows environment, the database is commonly deployed with the same name at all locations. A database administrator can now specify (in the new DBALIAS column in the table SYSIBM.LOCATIONS) multiple locations for a DB2 UDB for Linux, UNIX and Windows database that is deployed in multiple locations. As a result, a DB2 UDB for z/OS requester can now access multiple DB2 databases in the Linux, UNIX, and Windows environments that have the same name but different network addresses.
- A DB2 server is known in a network by its location name. Applications use the location name to identify an instance of a DB2 subsystem or a group of DB2 subsystems that share data. When you migrate two or more DB2 subsystems to a single data sharing group, you must consolidate multiple locations into a single location (migrating several subsystems to a single data sharing group is a complex task that might require other tools). After the locations are consolidated, you must change all applications that use the old location name to access the location name of the new data sharing group. When a large number of remote applications are deployed across a network, changing each application to use the new location name simultaneously is difficult. To support migration from multiple locations to a single location, DB2 lets you define eight location alias names for a DB2 subsystem or for a group of data-sharing DB2 subsystems. A location alias is another name that a requester can use to access a DB2 subsystem.
- You can also use a location alias with a TCP/IP port number to allow connections to a subset of data sharing members from DRDA requesters that connect to DB2 UDB for z/OS through TCP/IP.
- A database administrator can now override the automatic TCP/IP workload balancing function in a data sharing environment by setting up rows in the new SYSIBM.IPLIST table at a requester in conjunction with defining location aliases at the server. With the new SYSIBM.IPLIST table, a database administrator can define a specific member or a subset of members in a data sharing group. With this new service, applications can route requests by using a name that is different from the group location name.
- Version 8 also adds DRDA XA protocol support, which enables distributed transactions that implement the Java 2 Platform, Enterprise Edition (J2EE) Java Transaction Service (JTS), and Java Transaction API (JTA) specifications.

IBM z/OS Application Connectivity to DB2 for z/OS feature

z/OS Application Connectivity to DB2 for z/OS is a no-charge, optional feature of DB2 UDB for z/OS. This feature consists of a component known as the DB2 Universal Database Driver for z/OS, Java edition. This pure Java, type 4 JDBC driver is designed to deliver high performance and scalable remote connectivity for Java-based enterprise applications on z/OS to a remote DB2 for z/OS database server. The driver:

- Supports JDBC 2.0 and 3.0 specifications and Java Development Kit, Version 1.4 to deliver the maximum flexibility and performance that is required for enterprise applications

- Delivers robust connectivity to DB2 for z/OS and the WebSphere Application Server for z/OS
- Supports distributed transactions
- Allows custom Java applications that do not require an application server to run in a remote partition and connect to DB2 for z/OS

The DB2 Universal JDBC Driver is a single driver that includes JDBC type 2 and JDBC type 4 behavior and SQLJ support. When an application loads the DB2 Universal JDBC Driver, a single driver instance is loaded for type 2 and type 4 implementations. The application can make type 2 and type 4 connections by using this single driver instance. The DB2 Universal JDBC Driver supports the following JDBC and SQLJ functions:

- Most of the methods that are described in the JDBC 1.2 and JDBC 2.0 specifications and some of the methods that are described in the JDBC 3.0 specifications
- Connection pooling
- Global transactions that run on WebSphere Application Server, Version 5.0 and later
- Distributed transaction support that implements the Java 2 Platform, Enterprise Edition (J2EE) Java Transaction Service (JTS), and Java Transaction API (JTA) specifications

ODBC expands support for encoding schemes

ODBC now supports Unicode formats UTF-8 and UCS-2. In addition, a new ODBC initialization file keyword, CURRENTAPPENSCH, lets you specify the encoding scheme that you want the ODBC driver to use for input and output of host variable data, SQL statements, and all character string arguments of the ODBC application programming interfaces. You can specify one of the following encoding schemes: Unicode, EBCDIC, or ASCII.

Scalability and performance

In Version 8, DB2 breaks through limits and sets new heights for scalability and performance with:

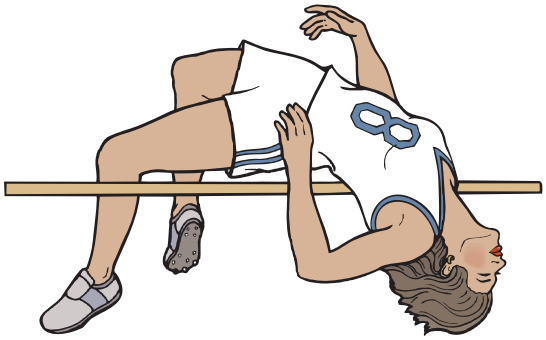

Figure 3. Version 8 of DB2 UDB for z/OS breaks through limits and sets new heights for scalability and performance.

64-bit virtual storage

Version 8 of DB2 UDB for z/OS, through exclusive integration with the IBM zSeries 800, 900, or the equivalent, now supports 64-bit virtual storage. The zSeries 64-bit architecture allows DB2 UDB for z/OS to move various storage areas above the 2^{31}-byte (2-GB) bar. The single large address space of up to 2^{64} bytes (16 exabytes) in Version 8 is 8 billion times larger than the address space that is available in Version 7. How big is 8 billion times larger? It is the difference between 5 centimeters and the distance from the earth to the moon. (See Figure 4.) Or, consider that if you start now with 2-GB of virtual storage and double it every year, you will reach the limit of 16 exabytes in 33 years. The immense size of the address space in Version 8, no matter how you visualize it, improves scalability and availability and gives you more flexibility to manage virtual storage.

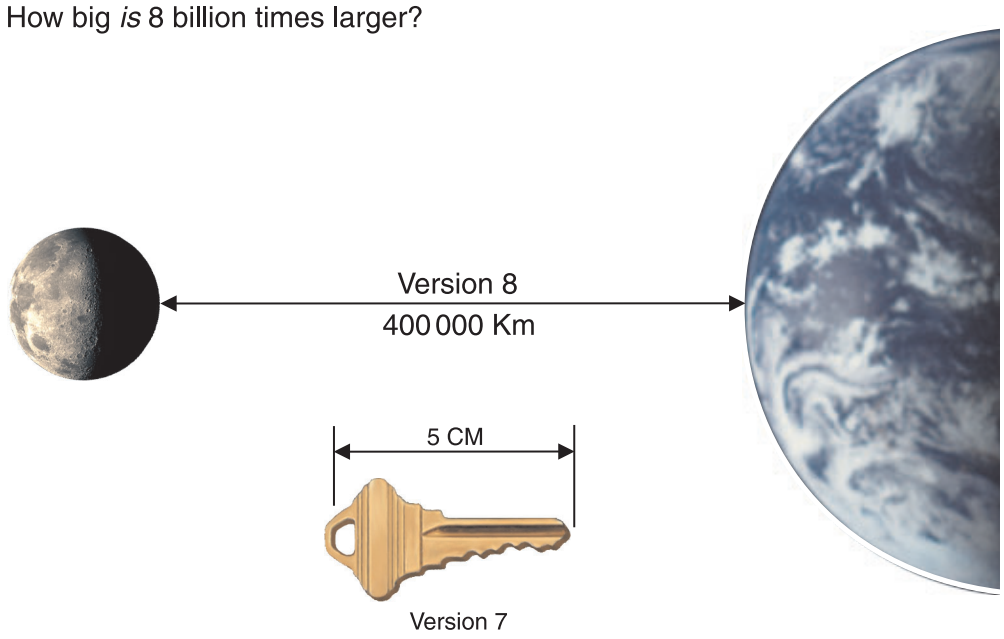

Figure 4. Relative size comparison of 31-bit virtual storage on OS/390 to 64-bit virtual storage on z/OS

Materialized query tables

Decision-support queries typically operate on large amounts of data (1 to 10 terabytes) to perform multiple joins and complex aggregation. To reduce the elapsed time of decision-support queries, Version 8 of DB2 UDB for z/OS supports materialized query tables. A materialized query table contains materialized data that is derived from one or more source tables that are specified by a fullselect in an SQL expression. (See Figure 5.) DB2 can then use the materialized data to answer a query more efficiently. A new clause on the CREATE TABLE statement lets you control whether a materialized query table is to be used automatically to answer queries. DB2 also lets you specify, within a query, whether you want DB2 to take advantage of available materialized query tables.

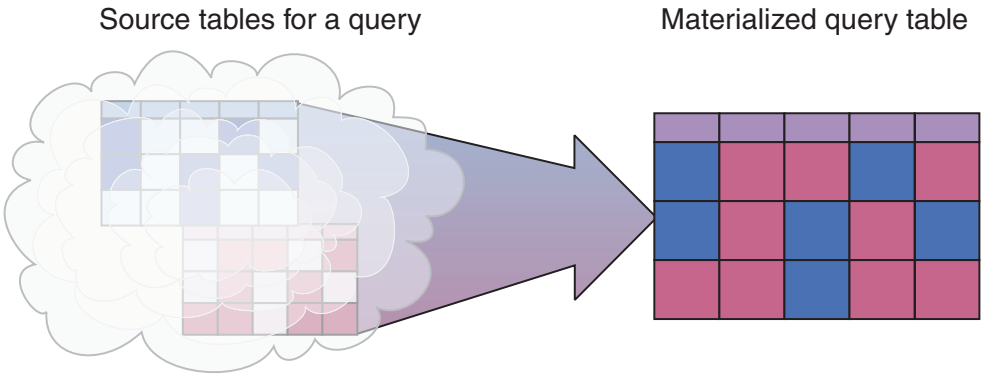

Figure 5. An illustration of a materialized query table that is derived from two source tables

Ability to use an index in more situations

Version 8 of DB2 UDB for z/OS gives you the flexibility to use an index in more situations for improved scalability and better performance. Improvements include:

- "Capability to index predicates that have mismatched data types"
- "Stored variable-length index keys"
- "Backward index scans for avoiding sort operations" on page 19

Capability to index predicates that have mismatched data types

In Version 8, you can now index many predicates that have mismatched data types; as a result, query performance improves. You can join tables on columns that have different data types and lengths, or you can provide a search value with a data type or length that does not match the definition of a column. For example, the C and C++ programming languages do not support the decimal data type, so programs that are written in these languages often use the floating-point data type for predicates on decimal columns. With Version 8 support for mismatched data types and lengths, predicates like those in the preceding example can now be stage 1 predicates.

Stored variable-length index keys

In Version 7, an index-only access path cannot be used for short VARCHAR host variables. In addition, varying-length columns that consist of VARCHAR and VARGRAPHIC data are varying-length columns in tables, but they are padded to their maximum length in index keys. Indexes that are padded require additional storage.

As Figure 6 on page 19 shows, Version 8, support for true varying-length keys in the index allows index-only access for indexes that have varying-length index keys and reduces storage requirements. In most cases, index keys in which

varying-length columns are not padded require less storage because only data is stored in the index. Version 8 also gives you the flexibility to create or alter indexes so that they have varying-length columns in the keys, and you can control whether DB2 pads those columns in the index.

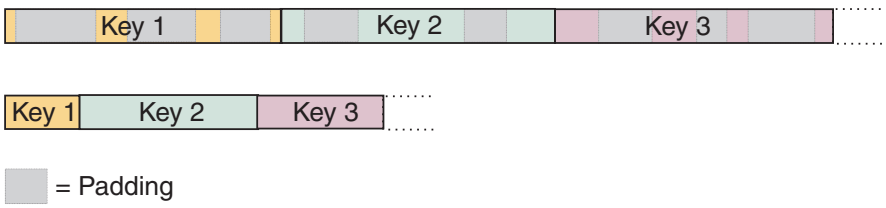

Figure 6. An illustration of true varying-length index keys in Version 8 of DB2 UDB for z/OS

Backward index scans for avoiding sort operations

Version 8 provides the capability for backward index scans, which lets DB2 avoid a sort. A sort is avoided when the DB2 optimizer determines that an ORDER BY clause can be satisfied by traversing an existing index in a backward direction. In prior releases, database designers would sometimes create a descending index for the sole purpose of avoiding a sort. Such descending indexes can now be deleted to improve performance because of the savings in index maintenance and reduced disk usage.

Additional distribution statistics for improved optimization

To run efficiently, data warehousing, data mining, and ad hoc query applications need statistics on columns that are in predicates, regardless of whether they are leading columns of an index. In addition, distribution statistics on non-leading index columns or non-indexed columns let DB2 make better access path decisions when data is asymmetrically distributed.

In Version 8, you can use the RUNSTATS utility to collect the following additional statistics:

- Frequency distributions for non-indexed columns or groups of columns
- Cardinality values for groups of non-indexed columns
- Least-frequently occurring values, most-frequently occurring values, or both, for any group of columns

Improved optimization for dynamic SQL

Several enhancements improve optimization for dynamic SQL:

- A new bind option, REOPT(ONCE), allows DB2 to determine and store only once at run time the access path for any dynamic SQL statement that contains variable values. DB2 uses the first set of input variables to determine the access path. The REOPT(ONCE) option can increase efficiency for dynamic SQL statements that run multiple times because DB2 does not reoptimize the access path each time a statement runs.
- A new clause of the EXPLAIN statement lets you obtain information about statements that have been stored in the dynamic statement cache. As a result, you can examine the current access path.

Improved trigger performance

In Version 8, DB2 requires fewer work files for processing conditional triggers. As a result, trigger performance improves.

More parallelism for sort operations

In Version 8, DB2 performs more parallel processing of some sort operations that are used in join processing. To ensure that parallel sort operations are cost-effective, DB2 uses a cost model to determine whether to perform a parallel sort.

Performance enhancements for star join qualified queries

The performance of a star join is critical to data warehousing applications in which the main database design is a star schema. The star join implementation in DB2 UDB for z/OS must handle a large number of work files. Because work files prior to Version 8 do not have indexes, sort-merge joins tend to be selected. As a result, the cost of sorting can be great in both time and the amount of space required.

Version 8 provides enhancements that improve the optimization and execution of a star join qualified query. Now, sparse indexes are supported on star join work files. As a result, the DB2 optimizer selects the access path based on the estimated costs of the access plans, which can boost query performance and can reduce the cost of sorting a larger number of work files.

Multiple fetches and inserts allowed within a single SQL statement

You can enhance the performance of your application programs by using multiple-row FETCH and INSERT statements to request that DB2 send multiple rows of data, at one time, to and from the database. Using these multiple-row statements in local applications results in fewer accesses of the database. Using these multiple-row statements in distributed applications results in fewer network operations and a significant improvement in performance. Figure 7 illustrates the difference between a series of single fetches and a single, multiple-row fetch operation.

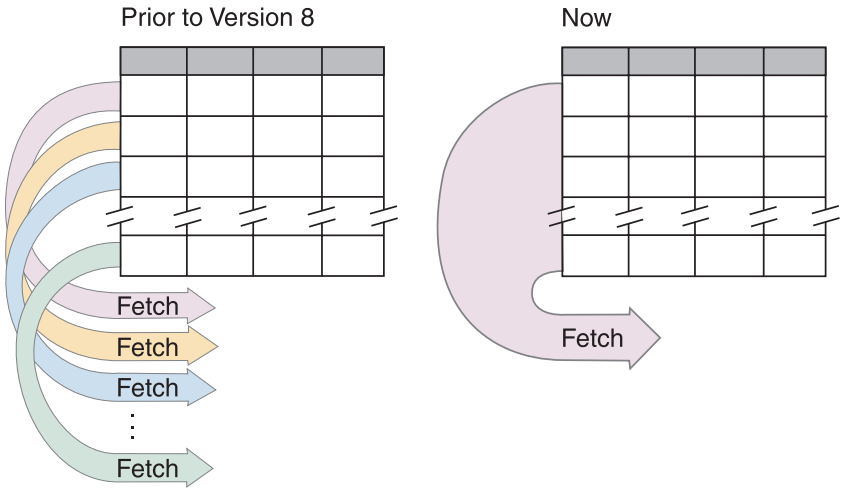

Figure 7. Before Version 8, you needed to use a series of single-row fetches to return many rows of data. Now, in Version 8, you can use multiple fetches within a single SQL statement to accomplish the same task more efficiently.

4096 partitions in a partitioned table space

As Figure 8 on page 21 shows, the maximum number of partitions in a partitioned table space is increased from 254 to 4096 partitions in Version 8. If you consider

that use of 254 partitions allows you to use one partition per day for 8 months, an increase to 4096 partitions extends those 8 months to 11 years.

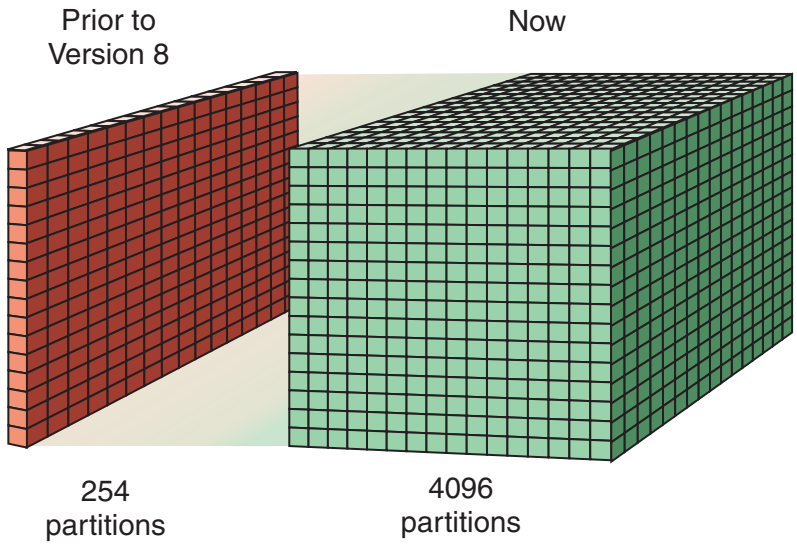

Figure 8. Comparison of 254 partitions to 4096 partitions in a partitioned table space

As a result of the increased number of partitions in a partitioned table space, the maximum size of a partitioned table is increased from 16 terabytes to 128 terabytes, as shown in Figure 9.

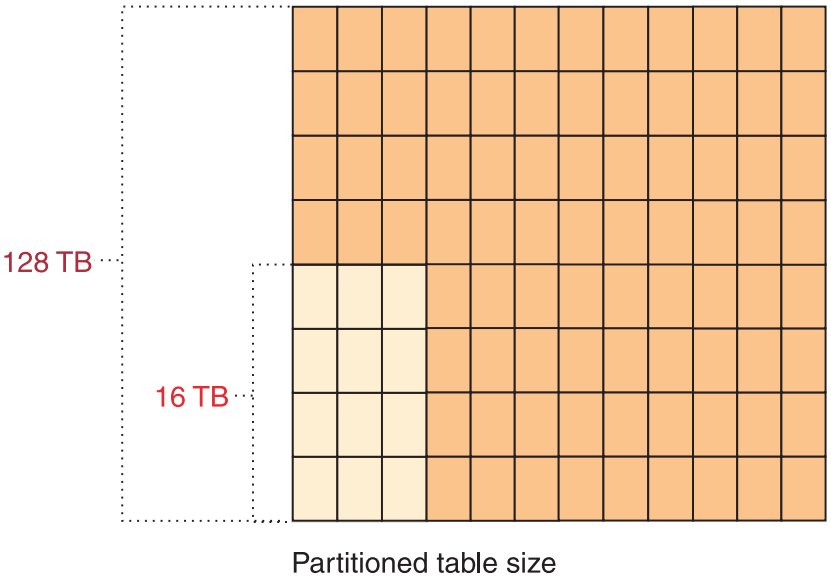

Figure 9. Comparison of a 16-terabyte partitioned table to a 128-terabyte partitioned table

The ability to have more partitions increases the granularity of your data and decreases the size of your data sets. For example, if one of your applications (prior to Version 8) stores daily data in separate partitions for one year, you would need up to 366 partitions. If you keep that data for five years, you would need 1830 partitions. To work around the 254-partition limit, you would need to either combine multiple days or weeks into partitions, or use simple table spaces and lose the benefit of parallel processing for queries. With Version 8, you reap the advantages of parallel processing for queries and utilities by using partitioned

table spaces: You have the flexibility to start with a minimum number of partitions and add more partitions as your needs change.

Greater resource control with stored procedures and user-defined functions

In Version 8, DB2 UDB for z/OS gives you greater control over resource utilization. You can now specify for each stored procedure or user-defined function the maximum number of failures (program exceptions or abnormal termination, for example) that are allowed before DB2 stops the routine. By specifying the most appropriate value for an individual routine, you can let some routines continue to be invoked for development and debugging, and stop other routines for maintenance before they cause problems in a production environment. Additional enhancements take advantage of z/OS Workload Manager functions that let System Resource Manager and Workload Manager determine appropriate resource utilization and recommend changes in the number of tasks that operate in a stored procedure address space. The stored procedure manager then adds or deletes tasks according to recommendations from Workload Manager.

Reduced overhead costs for data sharing workloads

Version 8 of DB2 UDB for z/OS provides two new batch processes that reduce the amount of traffic to and from the coupling facility when you run Version 1 Release 4 of z/OS and level 12 of the coupling facility. Two new commands let you write and register multiple pages to a group buffer pool, and read multiple pages from a group buffer pool for castout processing. You can expect the greatest performance benefits for data sharing workloads that update a large number of pages that belong to group-buffer-pool-dependent objects.

Reduced lock propagation in the coupling facility

Version 8 improves the performance of plans and packages that are bound with RELEASE(COMMIT) and increases the availability of your data because retained parent L-locks no longer lock an entire table or a table space when a DB2 member fails. You can now grant parent L-locks locally without invoking global contention processing. As a result, locking overhead that is caused by false contention is reduced.

Improved control for accounting aggregation

New e-business workloads that use the Recoverable Resource Manager Services attachment facility (RRSAF) or the distributed data facility (DDF) can generate enormous volumes of accounting records. To help reduce the volume of accounting records, Version 8 provides a new installation option (a subsystem parameter) that lets you accumulate data for RRSAF and DDF threads based on the following criteria:

- The user ID of the end user
- The transaction name of the end user
- The workstation name of the end user

Of course, some activities (detailed performance monitoring, for example) require detailed accounting data for RRSAF and DDF threads. With Version 8, you can dynamically alter the subsystem parameter to activate or deactivate data accumulation, which gives you the flexibility to meet your performance monitoring requirements.

Improved package-level accounting

New e-business workloads often use packages to issue SQL statements. With Version 8, package-level accounting contains additional detailed performance metrics that let you easily determine which packages were used to issue the SQL statements in your application workload. These metrics greatly simplify performance analysis activities for stored procedure applications, SQLJ applications, and DDF applications.

Reliability, availability, and serviceability

To keep your business competitive, the foundation of your e-business infrastructure must be reliable, available, and serviceable. Version 8 of DB2 UDB for z/OS continues to provide a strong foundation through the following enhancements:

Greater availability and flexibility with online schema evolution

In prior releases, your data and applications were unavailable during operations to alter schema definitions of table, table space, and index attributes, or to add, rotate, or rebalance partitions. Now, in Version 8, you can change schema definitions online for some table, table space, and index attributes or add, rotate, or rebalance partitions without losing availability of your data or applications. (See Figure 10 on page 24.) For example, you can change column types and lengths, add columns to an index, add, rotate, or rebalance partitions. Related enhancements provide more flexibility to change the clustering and partitioning of the data in a table.

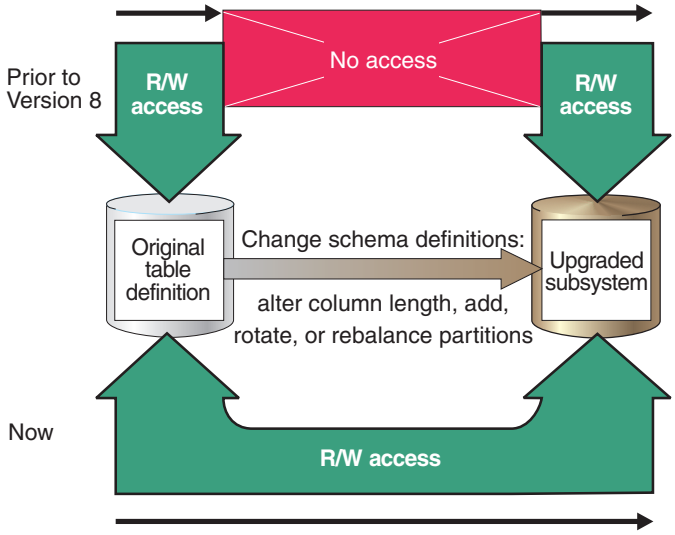

Figure 10. Greater availability with schema evolution

Change column types and lengths

You can change the data type of a table column. The new definition is applied immediately to all data that is in the associated table; when data rows are accessed, those rows are formatted according to the new definition.

Add columns to an index

You can now append columns to the end of an existing index. If you add a column to a table within the same unit of work in which the column is added to the index, the index is available for access immediately.

Add, rotate, or rebalance partitions dynamically

In Version 8, you can add, rotate, or rebalance partitions dynamically while maintaining availability of the partitions:

- You can add partitions at the end of a partitioned table space up to the maximum limit that is defined for the table space. The changes take effect immediately.
- You can reuse (or *rotate*) existing partitions for new data as old data is rolled away. With this support, you can, for example, keep 12 months of data continuously by using only 13 partitions. When you rotate partitions, you delete all the data rows in the oldest (first) partition and then specify a new high boundary for the table space. As a result, the oldest partition rotates to the last logical partition in sequence (the thirteenth partition, in this example) and is ready to hold new data.
- You can use the REBALANCE utility to rebalance partitions without causing a negative impact on availability.

Change the partitioning and clustering of the data in your tables

Version 8 gives you more flexibility for partitioning and clustering the data in your tables:

- You can create a partition without an index. As a result, you can improve performance by eliminating an unnecessary index and the accompanying overhead.
- You can drop a partitioning index or create a table without a partitioning index.

- You can define a clustering order that is different from the partitioning order. For example, if your data is partitioned by month, you can cluster by customer number within each partition.
- You can alter the clustering index.

Greater data availability with data-partitioned secondary indexes

Version 8 of DB2 lets you partition secondary indexes according to the partitioning scheme of the underlying data. *Secondary indexes* are nonpartitioning indexes of partitioned tables. Partitioned secondary indexes are referred to as *data-partitioned secondary indexes.* Data-partitioned secondary indexes can:

- Improve data availability during utility operations that operate at the partition level, such as REORG PART, LOAD PART, and RECOVER PART. For example, the BUILD2 phase and the accompanying outage are eliminated for REORG PART operations.
- Streamline and improve the performance of partition-level operations, such as rotating partitions. For example, the ALTER ROTATE PART operation performs a mass delete without accessing and deleting individual keys.
- Allow secondary indexes to benefit from strategies that reduce overhead costs that are incurred by data sharing.

More flexibility and faster recovery with system-level point-in-time recovery

Enhancements to system-level point-in-time recovery for DB2 provide improved usability, more flexibility, and faster recovery times. As Figure 11 on page 26 shows, you can now recover your data to any point in time, regardless of whether you have uncommitted units of work. As a result, data recovery time improves significantly for large DB2 subsystems that contain more than 30 000 objects, which means that the down time for the subsystem also decreases considerably. Two new utilities provide the vehicle for system-level point-in-time recovery:

- The BACKUP SYSTEM utility provides fast volume-level copies of DB2 databases and logs. It relies on new DFSMShsm™ services in z/OS Version 1 Release 5 that automatically keep track of the volumes that need to be copied. Using BACKUP SYSTEM is less disruptive than using the SET LOG SUSPEND command for copy procedures because DB2 does not suspend writing to the log. An advantage for data sharing is that BACKUP SYSTEM operates at the group scope, whereas SET LOG SUSPEND operates at the member scope.
- The RESTORE SYSTEM utility recovers a DB2 subsystem to an arbitrary point in time. RESTORE SYSTEM automatically handles any creates, drops, and LOG NO events that might have occurred between the backup and the recovery point in time.

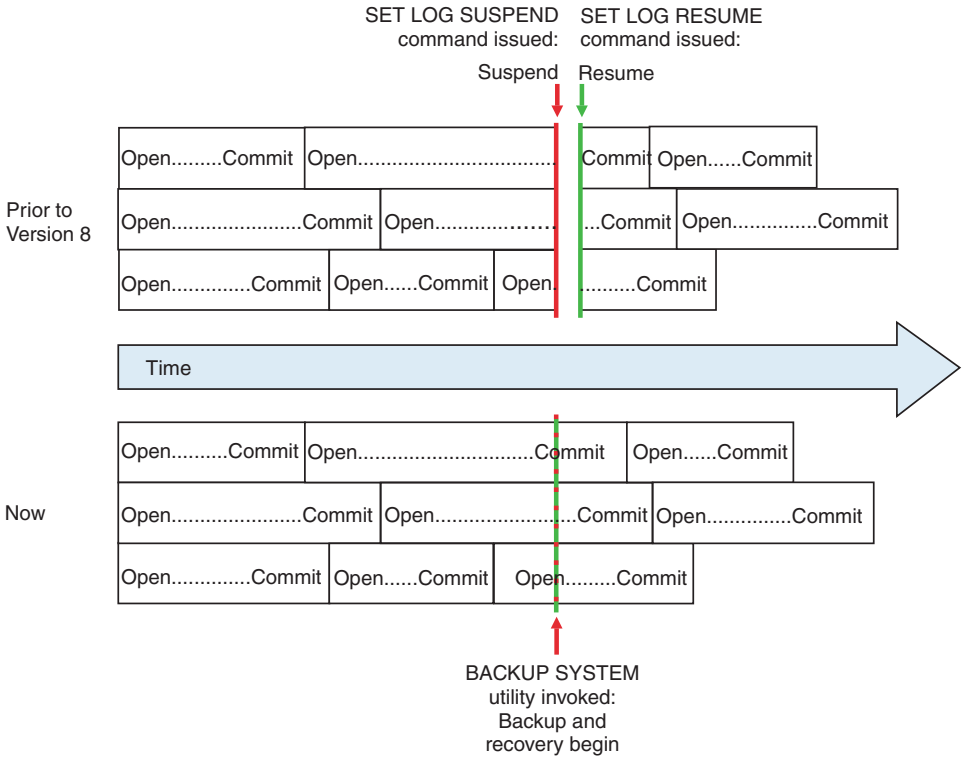

Figure 11. More flexibility and faster recovery with enhancements to system-level point-in-time recovery

Improved utility functions

Many of the Version 8 utility enhancements support other fundamental changes in Version 8 that have been discussed previously:

- Long names (see "Extended limits for names and SQL statements" on page 11)
- Unicode (see "Significant support for Unicode and long names in the DB2 catalog" on page 12)
- 64-bit virtual addressing ("64-bit virtual storage" on page 17)
- Data-partitioned secondary indexes (see "Greater data availability with data-partitioned secondary indexes" on page 25)
- System-level point-in-time recovery (see "More flexibility and faster recovery with system-level point-in-time recovery" on page 25)

Other utility changes include:

- "Online REORG utility"
- "LOAD and UNLOAD utilities" on page 27
- "RUNSTATS distribution statistics" on page 27
- "Autonomic restart" on page 27

Online REORG utility

In addition to REORG changes that support data-partitioned secondary indexes, as discussed in "Greater data availability with data-partitioned secondary indexes" on page 25, you can now:

- Specify the SHRLEVEL CHANGE option for a REORG TABLESPACE DISCARD operation.

- Reorganize DB2 catalog table spaces that have links if you specify SHRLEVEL REFERENCE.
- Specify the SCOPE PENDING keyword to reorganize only the partitions that are in a REORG-pending state (REORP) or an advisory REORG-pending (AREO) state for a specific table space or a partition range.

Online CHECK INDEX utility

Enhancements to the online CHECK INDEX utility in Version 8 increase the availability of your read-only data. New options let you specify:

- The number of seconds that the utility waits when draining a table space or an index
- The maximum number of retries that are to be attempted
- The minimum duration, in seconds, between retries

LOAD and UNLOAD utilities

Prior to Version 8, you could not load a delimited input file into DB2 or unload a delimited output file from DB2. A *delimited file* is a sequential file that contains row and column delimiters.

Now, in Version 8, you can use the LOAD utility to load into DB2 a delimited input file from another relational database. What's more, you do not need to write a program that converts the data into the correct format, or use INSERT processing and give up the performance advantages of the LOAD utility.

In addition, you now can use the UNLOAD utility to unload a delimited output file from DB2 to one or more files that are stored outside of DB2. You can then load the data into another DB2 database in a z/OS environment or on other operating systems, or import the data into an application in another relational database.

RUNSTATS distribution statistics

Skewed data distributions are responsible for a high proportion of performance problems with DB2 queries, especially in ad hoc queries. Symptoms of these problems include join sequences that are not optimal, too much synchronous I/O, and long response times. In addition, the ability of DB2 to make optimal decisions about table join order and table join methods can be weakened if the distribution of your data is asymmetrical, and you do not have distribution statistics on non-leading indexed columns or non-indexed columns.

Improvements to the RUNSTATS utility in Version 8 let you collect distribution statistics for non-leading indexed columns and non-indexed columns so that DB2 can use these statistics to select better access paths.

Autonomic restart

In Version 8, restarting utility jobs is easier because you no longer need to add the RESTART or RESTART(PHASE) parameters to a utility job. DB2 attempts to restart utility jobs that can be restarted online regardless of whether the RESTART keyword is specified.

Enhanced LPL recovery processing

DB2 inserts entries for pages that are logically in error in a logical page list (LPL). Version 8 of DB2 UDB for z/OS offers the following enhancements to LPL recovery to improve usability, serviceability, availability, and performance:

- Automatic recovery of LPL pages: To avoid manual intervention for LPL recovery through the START DATABASE command or the RECOVER utility, DB2, in most cases, automatically initiates an LPL recovery processor to recover pages as they are added to the LPL.
- Less-disruptive LPL recovery: The LPL recovery processor (by way of the START DATABASE command or the new automatic LPL recovery feature), makes a write claim instead of a drain on the object that is being recovered. As a result, good pages in the object are available to SQL users, and performance is improved because the claim is less disruptive than a drain.

Change more parameters online without recycling DB2

Version 7 provided support that let you change a set of subsystem parameters online without recycling DB2. Version 8 expands that set to include many more subsystem parameters that can be changed online. For example, you can now change authorization IDs for the system operators or the system administrator, change the sort pool size, and change the timeout limits for threads.

Increased maximum number of active and archive log data sets

In Version 8, an increase in the maximum number of active and archive log data sets per log copy increases recovery performance and scalability as the volume of your data increases.

The maximum number of archive log volumes that are recorded in the BSDS is increased from 1000 to 10 000 volumes per log copy. With a larger maximum number of archive log volumes per log copy, you can avoid taking frequent image copies.

The maximum number of active log data sets is increased from 31 to 93 per log copy. Recovery performance is improved because reading active logs is much faster than reading archive logs, and the additional active log capacity can decrease the necessity to read archive logs in some cases. Also, increasing the number of active logs can help avoid a full log condition if the offload task stalls or temporarily falls behind.

Autonomic space allocation

Version 8 provides space allocation enhancements to improve performance, increase data availability, and limit the occurrence of outages caused by lack of space. DB2 can now calculate the amount of space to allocate to secondary extents by using a sliding-scale algorithm. The first 127 extents are allocated in increasing size, and the remaining extents are allocated based on the initial size of the data set. This approach has several advantages:

- It minimizes the potential for wasted space by increasing the size of secondary extents gradually.
- It prevents very large allocations for the remaining extents, which would likely cause fragmentation.
- It does not require users to specify secondary space allocation values when creating and altering table spaces and index spaces.
- It allows you, in theory, to always reach the maximum data set size without running out of secondary extents.